Workshop on
ML Security

17/09/25

Verifiable Federated Learning with incremental ZKP

Aleksei Korneev
University of Lille, Inria Lille

Funded by
the European Union

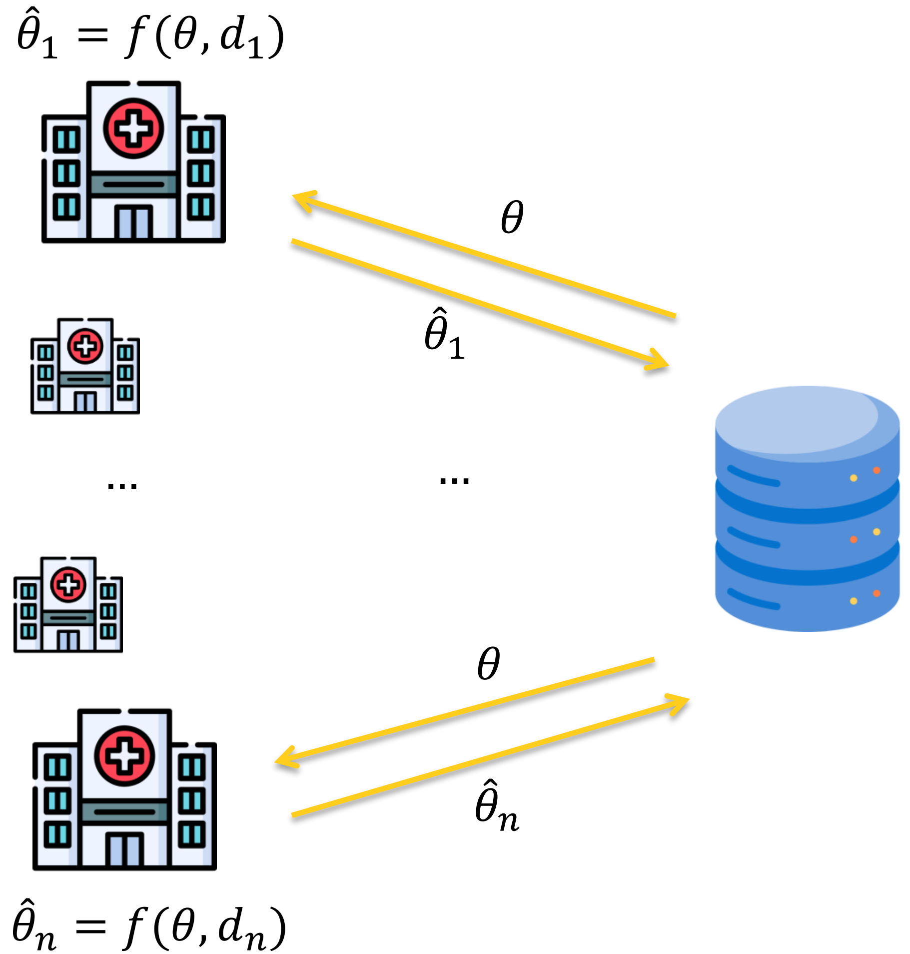
Content

- Introduction & Background
  - Federated Learning
  - Adversarial attacks in FL
  - Verifiable FL
- Verification with ZKP
  - Schnorr’s Sigma-Protocol
  - General purpose ZKP
- ZKP for FL
  - Verifiable FL with Incremental ZKP

Content

Federated learning

Data Owners perform computations using private data

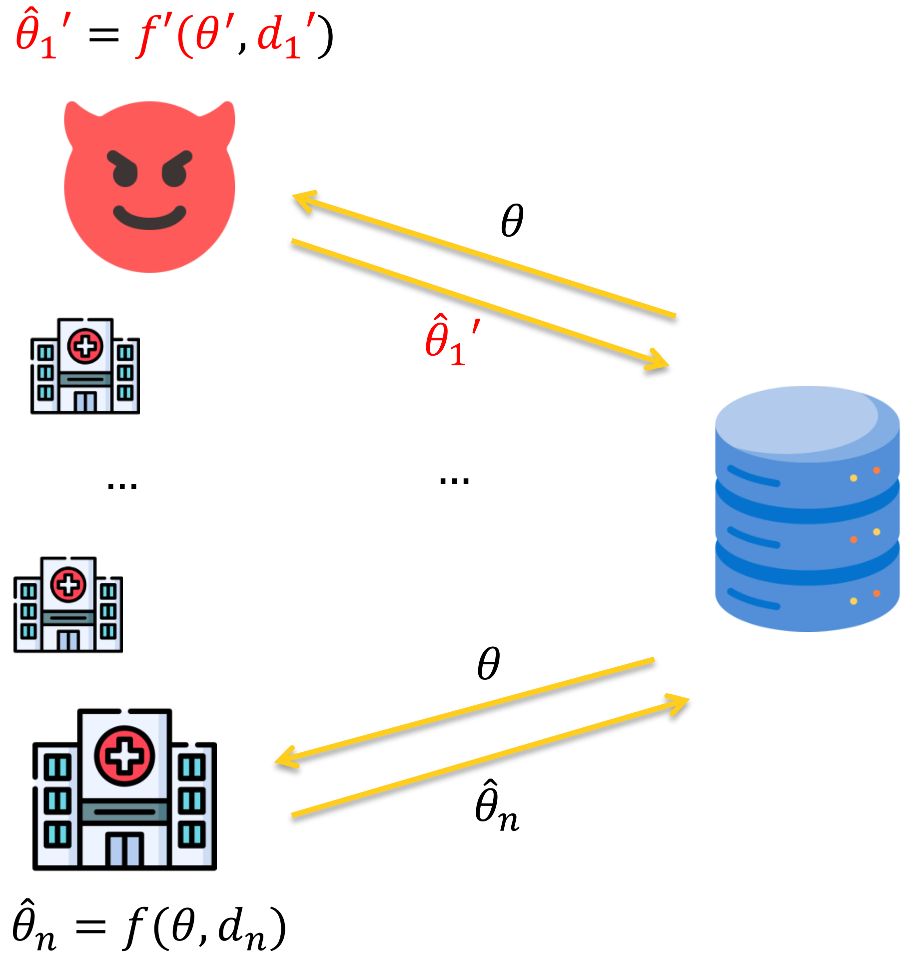


Aggregator coordinates the training

images: Flaticon.com Devil icons created by Saepul Nahwan - Flaticon
Database icons created by Freepik - Flaticon

Adversarial attacks in FL: malicious DO

Data Owners perform computations using private data

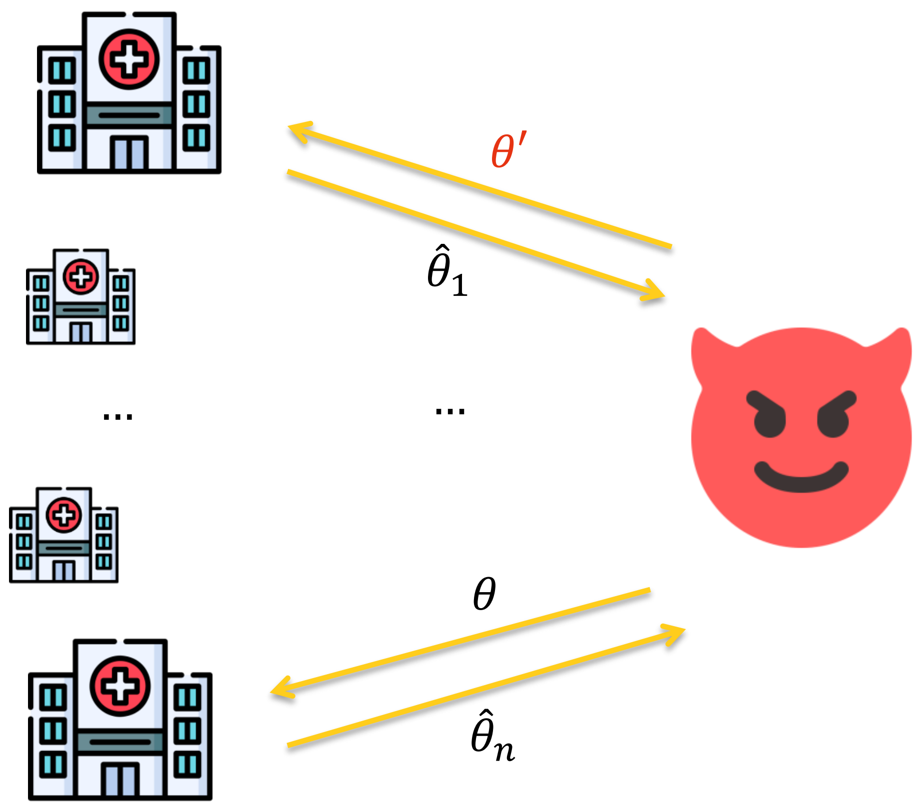


Aggregator coordinates the training

Adversarial attacks in FL: malicious aggregator

$\hat{\theta}_1 = f(\theta', d_1)$

Data Owners perform computations using private data

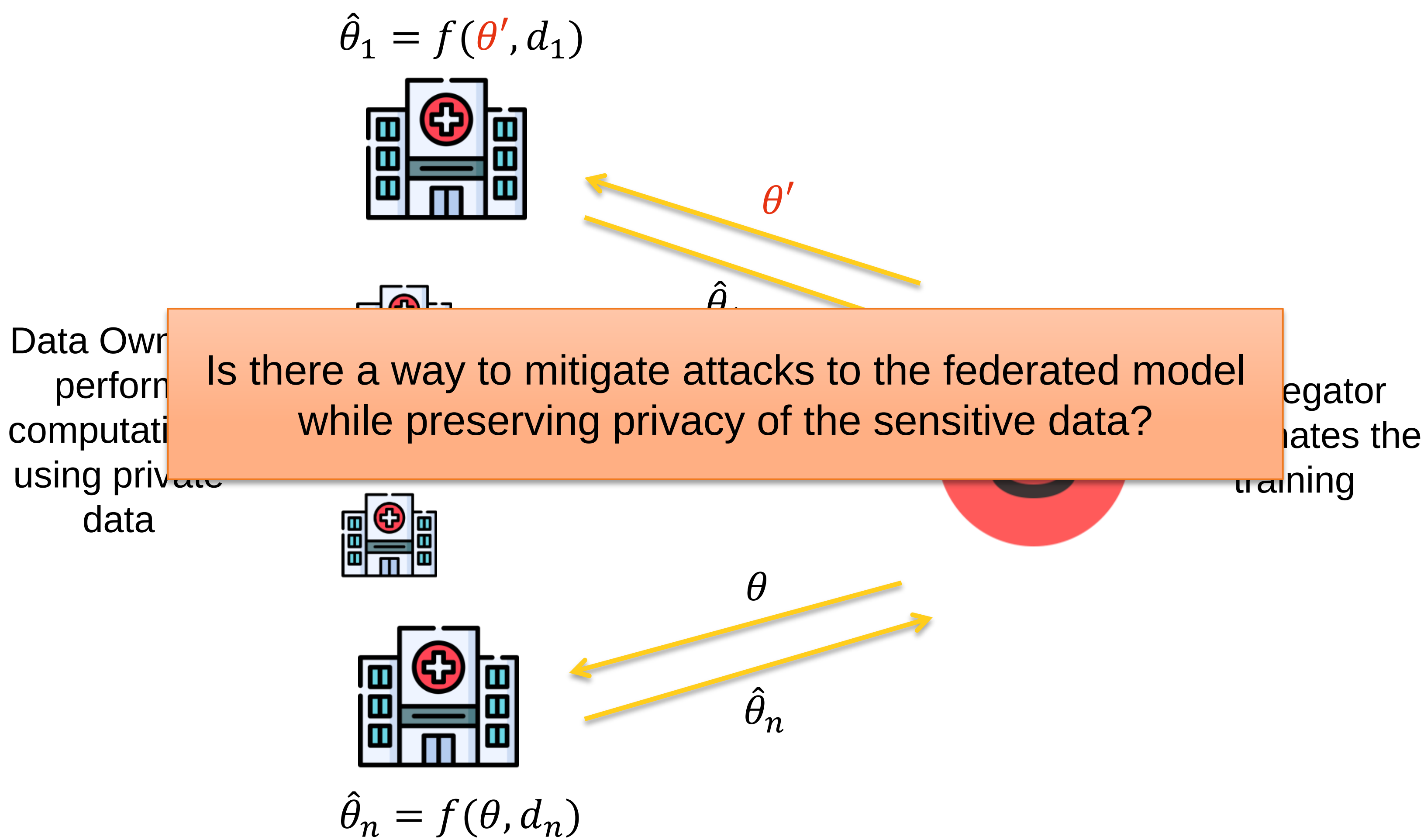


Aggregator coordinates the training

$\hat{\theta}_n = f(\theta, d_n)$

images: Flaticon.com Devil icons created by Saepul Nahwan - Flaticon
Database icons created by Freepik - Flaticon

Adversarial attacks in FL: malicious server



Is there a way to mitigate attacks to the federated model while preserving privacy of the sensitive data?

images: Flaticon.com Devil icons created by Saepul Nahwan - Flaticon
Database icons created by Freepik - Flaticon

Verifiable FL

Definition (Verifiable FL). FL is **verifiable** if selected parties are able to verify that the tasks of all participants are correctly performed without deviation.

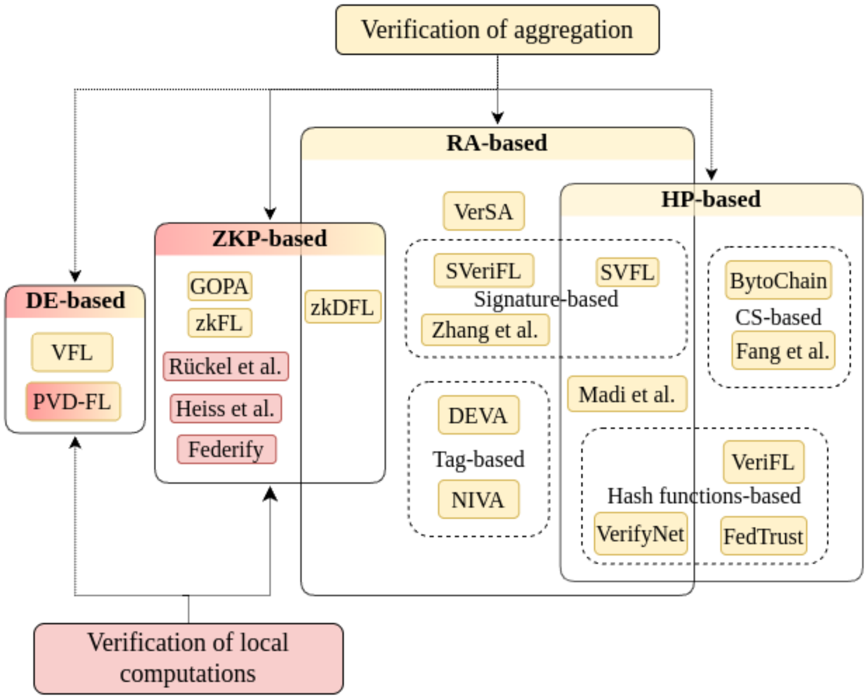
- All participants, i.e. both data owners and server(s)
- Attacks mitigation: no free-riders, no model-poisoning, no data-poisoning*

Verifiable FL

Definition (Verifiable FL). FL is **verifiable** if selected parties are able to verify that the tasks of all participants are correctly performed without deviation.

- All participants, i.e. both data owners and server(s)
- Attacks mitigation: no free-riders, no model-poisoning, no data-poisoning*

A survey on Verifiable Cross-Silo FL



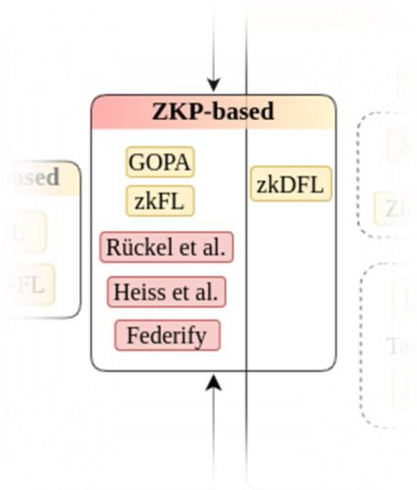
A Survey on Verifiable Cross-Silo Federated Learning

A Korneev, J Ramon - Transactions on Machine Learning Research …, 2025 - hal.science

Federated Learning (FL) is a widespread approach that allows training machine learning (ML) models with data distributed across multiple storage units. In cross-silo FL, which often …

Verification with ZKP

Zero-Knowledge Proof (ZKP) is a method by which one party can prove to another party the validity of a statement without revealing the statement itself.



- Arbitrary* computations
- Better complexities
  - To verify the result is cheaper than to compute
- Flexible proofs: one can reinforce the proof with additional info (noise, fairness, other local model's properties, …)

Content

- Introduction & Background
  - Federated Learning
  - Adversarial attacks in FL
  - Verifiable FL
- **Verification with ZKP**
  - **Schnorr's Sigma-Protocol**
  - **General purpose ZKP**
- ZKP for FL
  - Verifiable FL with Incremental ZKP

Schnorr's Sigma-Protocol

Prover: y, g, w

$y = g^w$

Verifier: y, g

1. Prover computes $C = g^a$ (a is chosen randomly)

$\xrightarrow{\quad C \quad}$

$\xleftarrow{\quad \mathrm{r} \quad}$

2. Verifier generates a random challenge r

3. Prover computes $z = a + rw$

$\xrightarrow{\quad z \quad}$

4. Verifier checks that $g^z = C \cdot y^r$

$$g^{a+rw} = g^a \cdot (g^w)^r$$

Schnorr's Sigma-Protocol

Prover: y, g, w

$y = g^w$

Verifier: y, g

1. Prover computes $C = g^a$

C

2. Prover generates a challenge $\mathrm{r} = H(C)$

3. Prover computes $z = a + rw$

z

4. Verifier checks that $g^z = C \cdot y^{H(C)}$

images: Flaticon.com

Schnorr's Sigma-Protocol

$y = g^w$

Prover

1. Prover computes $C = g^a$

2. Prover generates a challenge $\mathrm{r} = H(C)$

3. Prover computes $z = a + rw$

Proving algorithm

C

z

Proof

Verifier

4. Verifier checks that $g^z = C \cdot y^r$

Verification algorithm

images: Flaticon.com

Schnorr's Sigma-Protocol

Public I/O Witness

$$y = g^w$$

Prover

Verifier

Proving algorithm

Proof

Verification algorithm

images: Flaticon.com

General purpose ZKP

Public I/O　　　Witness

$$y = f(x, w)$$

Prover

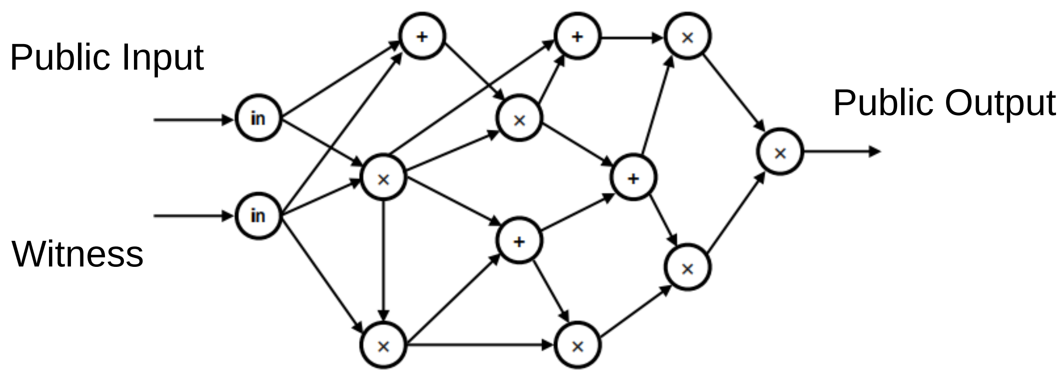
Verifier

Proving algorithm

Proof

Verification algorithm

f could be represented as a circuit:



images: Flaticon.com

General purpose ZKP

Prover

$$y = f(x, w)$$

Verifier

$$\pi \leftarrow Prove(f, x, y, w) \xrightarrow{\pi, y} 1/0 \leftarrow Verify(f, x, y, \pi)$$

images: Flaticon.com

General purpose ZKP with preprocessing

Prover

$y = f(x, w)$

Verifier

$pk \leftarrow Preproc(f)$
$\pi \leftarrow Prove(pk, x, y, w)$

$\xrightarrow{\pi, y}$

$vk \leftarrow Preproc(f)$
$1/0 \leftarrow Verify(vk, x, y, \pi)$

images: Flaticon.com

General purpose ZKP

Table 3: Asymptotic comparison of ZKP schemes with logarithmic and constant proof size complexity. C is the computation expressed as a circuit, $|C|$ is the number of gates in the circuit, $|N|$ is the length of inputs.

Scheme	Parameters size	Proving	Verification	Proof Size
Dory (Lee, 2021)	$O(\|C\|)$	$O(\|C\|)$	$O(\log\|C\|)$	$O(\log\|C\|)$
Gemini (space-efficient) (Bootle et al., 2022)	$O(\|C\|)$	$O(\|C\|\log^2\|C\|)$	$O(\log\|C\|)$	$O(\log\|C\|)$
Gemini (time-efficient) (Bootle et al., 2022)	$O(\|C\|)$	$O(\|C\|)$	$O(\log\|C\|)$	$O(\log\|C\|)$
SuperSonic (Bünz et al., 2020)	$O(1)$	$O(\|C\|\log\|C\|)$	$O(\log\|C\|)$	$O(\log\|C\|)$
DARK-fix (Arun et al., 2023)	$O(1)$	$O(\|C\|\log\|C\|)$	$O(\log\|C\|)$	$O(\log\|C\|)$
BCCGP (Bootle et al., 2016)	$O(\|C\|)$	$O(\|C\|)$	$O(\|C\|)$	$O(\log\|C\|)$
Bulletproofs (Bunz et al., 2018)	$O(\|C\|)$	$O(\|C\|)$	$O(\|C\|)$	$O(\log\|C\|)$
Compressed Σ-protocol (Attema & Cramer, 2020)	$O(\|C\|)$	$O(\|C\|)$	$O(\|N\|)$	$O(\log(\|C\|))$
Groth16 (Groth, 2016)	$O(\|C\|)$	$O(\|C\|\log\|C\|)$	$O(\|N\|)$	$O(1)$
Sonic (Maller et al., 2019)	$O(\|C\|)$	$O(\|C\|\log\|C\|)$	$O(N)$	$O(1)$
GGPR (Gennaro et al., 2013)	$O(\|C\|)$	$O(\|C\|\log\|C\|)$	$O(\|N\|)$	$O(1)$
Pinochio (Parno et al., 2013)	$O(\|C\|)$	$O(\|C\|\log\|C\|)$	$O(\|N\|)$	$O(1)$
PLONK (Gabizon et al., 2019)	$O(\|C\|)$	$O(\|C\|\log\|C\|)$	$O(\|N\|)$	$O(1)$
vnTinyRAM (Ben-Sasson et al., 2014)	$O(\|C\|\log\|C\|)$	$O(\|C\|\log^2\|C\|)$	$O(\|N\|)$	$O(1)$
Mirage (Kosba et al., 2020)	$O(\|C\|)$	$O(\|C\|\log\|C\|)$	$O(\|N\|)$	$O(1)$
Behemoth (Seres & Burcsi, 2023)	$O(\|C\|)$	$O(\|C\|^3\log\|C\|)$	$O(\|N\|)$	$O(1)$
Dew (Arun et al., 2023)	$O(1)$	$O(\|C\|^2)$	$O(\log\|C\|)$	$O(1)$

Content

- Introduction & Background
  - Federated Learning
  - Adversarial attacks in FL
  - Verifiable FL
- Verification with ZKP
  - Schnorr’s Sigma-Protocol
  - General purpose ZKP
- **ZKP for FL**
  - **Verifiable FL with Incremental ZKP**

ZKP for practical FL

- ML models often contain many parameters, one can have a circuit C with $> 10^6$ gates
  - Proof size: at most $log(C)$
  - Verification : at most $log(C)$
  - Prover's storage: at most $log(C)$

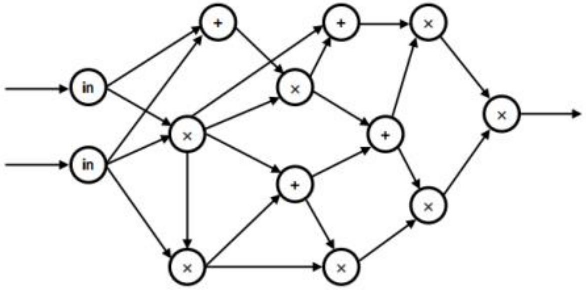
Scheme	Parameters size	Proving	Verification	Proof Size
SuperSonic (Bünz et al., 2020)	$O(1)$	$O(\|C\|\log\|C\|)$	$O(\log\|C\|)$	$O(\log\|C\|)$
DARK-fix (Arun et al., 2023)	$O(1)$	$O(\|C\|\log\|C\|)$	$O(\log\|C\|)$	$O(\log\|C\|)$

  - Prover's runtime memory: at most $log(C)$

$\downarrow$

Existing general-purpose ZKP do not fit the requirements for practical FL.

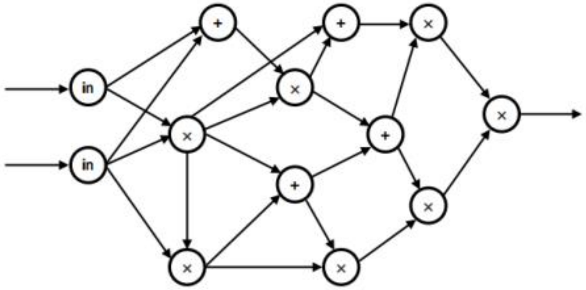
ZKP for partitioned circuits



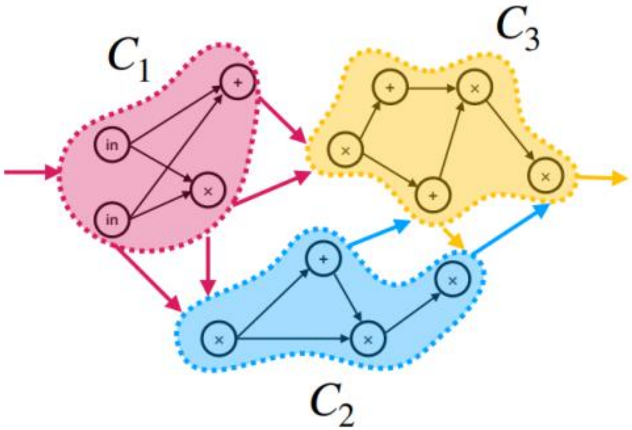
(**a**) Unpartitioned circuit C.

Rosenberg et al., Hekaton: Horizontally-Scalable zkSNARKs via Proof Aggregation.,

ZKP for partitioned circuits



(a) Unpartitioned circuit C.



(b) Partitioning C into 3 subcircuits.

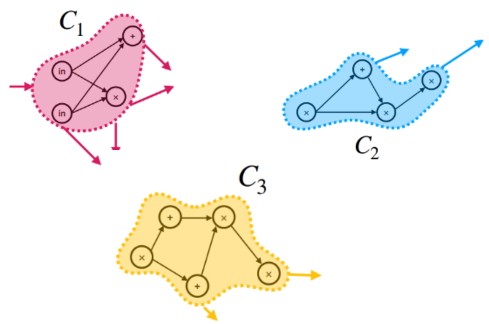
Rosenberg et al., Hekaton: Horizontally-Scalable zkSNARKs via Proof Aggregation.,

ZKP for partitioned circuits

Prover

$$y = f(x, w)$$

Verifier



$$pk_1 \leftarrow Preproc(f_1)$$
$$pk_2 \leftarrow Preproc(f_2)$$
$$pk_3 \leftarrow Preproc(f_3)$$

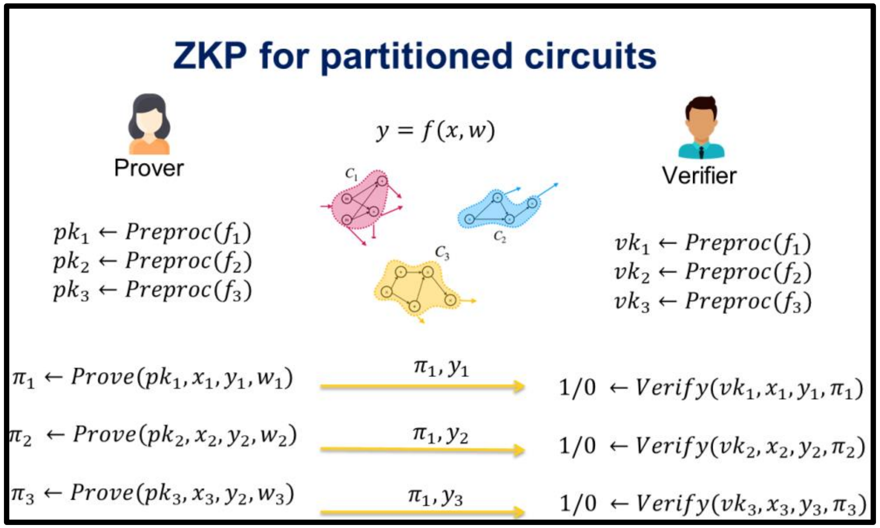
$$vk_1 \leftarrow Preproc(f_1)$$
$$vk_2 \leftarrow Preproc(f_2)$$
$$vk_3 \leftarrow Preproc(f_3)$$

$$\pi_1 \leftarrow Prove(pk_1, x_1, y_1, w_1) \xrightarrow{\pi_1, y_1} 1/0 \leftarrow Verify(vk_1, x_1, y_1, \pi_1)$$

$$\pi_2 \leftarrow Prove(pk_2, x_2, y_2, w_2) \xrightarrow{\pi_2, y_2} 1/0 \leftarrow Verify(vk_2, x_2, y_2, \pi_2)$$

$$\pi_3 \leftarrow Prove(pk_3, x_3, y_2, w_3) \xrightarrow{\pi_3, y_3} 1/0 \leftarrow Verify(vk_3, x_3, y_3, \pi_3)$$

ZKP for partitioned circuits



New problems to solve:

- Proofs are independent: no guarantee that the same secret values are used
- Intermediate values are revealed
- Communication and computational costs are higher

Incremental ZKP

Prover

$$y = f(x, w)$$

Verifier

$pk \leftarrow Preproc(\hat{f})$

$vk \leftarrow Preproc(\hat{f})$

$$\pi_1 \leftarrow Prove(pk, x_1, w_1, \_)$$
$$\pi_2 \leftarrow Prove(pk, x_2, w_2, \boldsymbol{\pi_1})$$
$$\pi_3 \leftarrow Prove(pk, x_3, w_3, \boldsymbol{\pi_2})$$

$\xrightarrow{\pi_3, y_3}$

$$1/0 \leftarrow Verify(vk, x_3, y_3, \pi_3)$$

- Is there a ZKP scheme to compute proofs recursively ?
- How to implement the function $\hat{f}$?

Nova

Paper 2021/370

Nova: Recursive Zero-Knowledge Arguments from Folding Schemes

Abhiram Kothapalli, Carnegie Mellon University
Srinath Setty, Microsoft Research
Ioanna Tzialla, New York University

Abstract

We introduce a new approach to realize incrementally verifiable computation (IVC), in which the prover recursively proves the correct execution of incremental computations of the form $y = F^{(\ell)}(x)$, where F is a (potentially non-deterministic) computation, x is the input, y is the output, and $\ell > 0$. Unlike prior approaches to realize IVC, our approach avoids succinct non-interactive arguments of knowledge (SNARKs) entirely and arguments of knowledge in general. Instead, we introduce and employ folding schemes, a weaker, simpler, and more efficiently-realizable primitive, which reduces the task of checking two instances in some relation to the task of checking a single

Metadata

Available format(s)

PDF

Category

Foundations

Publication info

A major revision of an IACR publication in CRYPTO 2022

Keywords

incrementally verifiable computation
zero knowledge arguments
recursive proof composition

Contact author(s)

Nova and more...

Paper 2022/1758

SuperNova: Proving universal machine executions without universal circuits

Abhiram Kothapalli, Carnegie Mellon University
Srinath Setty, Microsoft Research

Abstract

This paper intro
incrementally p
stateful machin
distinguishing a
program is prop
instruction invo
works that emp
is proportional
supported instr
of the supporte
instruction set w
achieves its cos
system, and lev
manner. We for

Metadata

Available form

PDF

Category

Paper 2023/573

HyperNova: Recursive arguments for customizable constraint systems

Abhiram Kothapalli, Carnegie Mellon University
Srinath Setty, Microsoft Research

Abstract

We introduce HyperNova, a new recursive argument for proving incremental

Metadata

Available format(s)

PDF

Category

Foundations

Publication info

Paper 2023/620

ProtoStar: Generic Efficient Accumulation/Folding for Special So Protocols

Benedikt Bünz, Espresso Systems
Binyi Chen, Espresso Systems

Abstract

Accumulation is a simple yet powerful primitive that enables incre
verifiable computation (IVC) without the need for recursive SNAR
a generic, efficient accumulation (or folding) scheme for any $(2k$
special-sound protocol with a verifier that checks ℓ degree-d equ
accumulation verifier only performs $k+2$ elliptic curve multiplica
$k+d+O(1)$ field/hash operations. Using the compiler from BCL
21), this enables building efficient IVC schemes where the recursiv
depends on the number of rounds and the verifier degree of the
special-sound protocol but not the proof size or the verifier time.
generic accumulation compiler to build ProtoStar. ProtoStar is a n
IVC scheme for Plonk that supports high-degree gates and (vecto

Paper 2023/1106

ProtoGalaxy: Efficient ProtoStar-style folding of multiple instances

Liam Eagen, Blockstream Research, Zeta Function Technologies
Ariel Gabizon, Zeta Function Technologies

Abstract

We continue the recent line of work on folding schemes. Building on ideas from ProtoStar [BC23] we construct a folding scheme where the recursive verifier's ``marginal work'', beyond linearly combining witness commitments, consists only of a logarithmic number of field operations and a constant number of hashes. Moreover, our folding scheme performs well when \emph{folding multiple instances at one step}, in which case the marginal number of verifier field operations per instance becomes constant, assuming constant degree gates.

Note: indexing typo

Metadata

Available format(s)

PDF

Category

Cryptographic protocols

Publication info

Preprint.

Keywords

SNARKs Folding Schemes

Contact author(s)

liameagen @ protonmail com
ariel gabizon @ gmail com

Inria

Universal Function

$Function\ \hat{f}\big(([op], vk, x_{i-1}, x_i), (w, \pi)\big)$:

$if\, op_i == 1$:

$x_i = f_1(x_{i-1}, w)$

$if\, op_i == 2$:

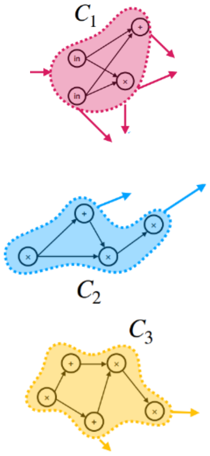
$x_i = f_2(x_{i-1}, w)$

$if\, op_i == 3$:

$x_i = f_3(x_{i-1}, w)$

$1 == Verify(vk, op_{i-1}, x_{i-1}, x_i, \pi)$

$Output\quad y$



Universal Function

$Function\ \hat{f}\big(([op], vk, x_{i-1}, x_i), (w, \pi)\big)$:

$if\, op_i == 1:$

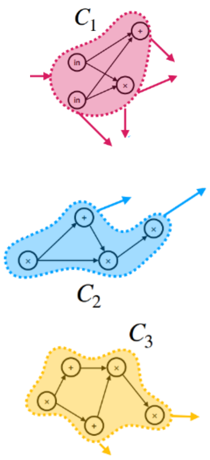
$x_i = f_1(x_{i-1}, w)$

$\ldots$

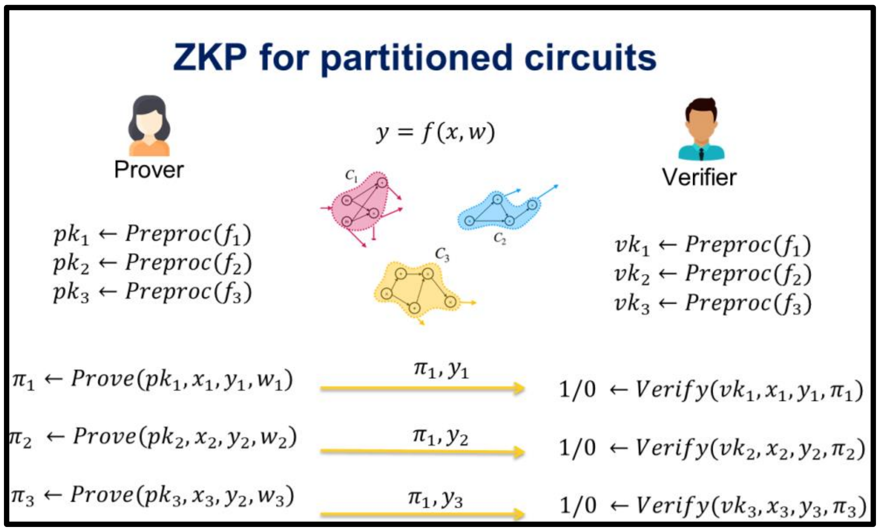
$x_i = f_3(x_{i-1}, w)$

$1 == Verify(vk, op_{i-1}, x_{i-1}, x_i, \pi)$

$Output\ \ y$



ZKP for partitioned circuits



New problems to solve:

- Proofs are independent
- ☑ Intermediate values are not revealed
- ☑ Communication and computational costs are reduced

Checkbox icons created by inkubators - Flaticon

Universal Function with shared memory

$Function\ \hat{f}\big(([op], vk, x_{i-1}, x_i, mem\_hash), (\pi, mem)\big)$:

$w = read(mem)$
$mem\_hash == H(mem)$

$if\, op_i == 1:$
$\quad x_i = f_1(x_{i-1}, w)$
$\quad ...$
$\quad x_i = f_3(x_{i-1}, w)$

$1 == Verify(vk, op_{i-1}, x_{i-1}, x_i, \pi)$

$Output\ \ y$

Verifiable FL via Incremental ZKP: example

Data owner

Aggregator (verifier)

$pk \leftarrow Preproc(\hat{f})$

$vk \leftarrow Preproc(\hat{f})$

$\xleftarrow{\theta}$

$$y_1 = \hat{f}(\theta, d)$$
$$\pi_1 \leftarrow Prove(pk, \theta, y_1, d, \_)$$
$$y_2 = \hat{f}(y_1, d)$$
$$\pi_2 \leftarrow Prove(pk, y_1, y_2, d, \pi_1)$$
$$\ldots$$
$$\hat{\theta} = \hat{f}(y_{n-1}, d)$$
$$\pi_n \leftarrow Prove\big(pk, y_{n-1}, \hat{\theta}, d, \pi_{n-1}\big) \xrightarrow{\hat{\theta}, \pi_n} 1/0 \leftarrow Verify(vk, \theta, \hat{\theta}, \pi_n)$$

Inria

Comparison with other approaches

Verifiable FL protocol	Proof size	DO's proving cost	DO's runtime memory	Verification cost	ZKP scheme
Rückel et al.	$O(1)$	$O(C\log(C))$	$O(C)$	$O(1)$	Groth16
Heiss et al.	$O(1)$	$O(C\log(C))$	$O(C)$	$O(1)$	Groth16
Federify et al.	$O(1)$	$O(C\log(C))$	$O(C)$	$O(1)$	Groth16
Our protocol	$O(\log(C))$	$O(C\log(C))$	$O(C')$	$O(\log(C))$	Nova* + Spartan

Table 1: A comparison of asymptotic complexity of ZKP-based FL protocols, where
C – the size of a circuit that represents DO's calculations,
C' - the size of the largest subcircuit.

Thank you for the attention!